

48  
185 pts.

# ARUN

Enciclopedia Práctica del Spectrum



Nueva Lente/Ingelek





# ENSAMBLADORES



¡echamos un vistazo al mundo informático actual, nos daremos cuenta de la gran cantidad de lenguajes que nos rodean, empezando por diferentes dialectos del BASIC, uno por cada máquina (o incluso algunos ordenadores tienen posibilidad de incorporar diferentes versiones de BASIC por software, como es el caso del SPECTRUM), y terminando por las diferentes versiones de otros lenguajes con nombres de claves secretas de espías.

Si profundizamos un poco más en la inmensa «Torre de Babel» informática nos percataremos que cada lenguaje ha sido creado para un uso concreto: BASIC, para aprender; DDBOL, para el manejo de gran cantidad de información; LOGO, para los niños; PASCAL, el superestructurado FORTRAN, para los cálculos matemáticos... Sin embargo, por muchos lenguajes que nos rodeen y nos amenacen a todos, en el fondo, y para la máquina, se reducen a uno: el lenguaje máquina. Cuando estamos trabajando con un lenguaje de alto nivel, en el momento de ejecutarlo es necesario que el «micro» lo traduzca a código máquina, que es lo único que entiende nuestra Z-80. El proceso de traducción, sobre todo si el lenguaje que estamos utilizando es un intérprete, es el que retrasa todo programa no siendo lo suficientemente rápido para ciertas aplicaciones. Entonces ¿por qué no nos animamos a programar en código máquina?

Si ya nos hemos recuperado del susto por la anterior frase desafiante, intentaremos ver de una forma objetiva si es conveniente o no realizar un programa en código máquina.

Entre las ventajas que nos aporta, lo más llamativo es su velocidad. Un programa en este lenguaje es evidente que no necesita traducción porque ya le hablamos a la máquina en el lenguaje que le gusta, y por tanto tarda bastante menos en ejecutar un programa (en algunos casos la velocidad es 60 veces superior a la del BASIC). Otra de las ventajas del lenguaje máquina es la poca memoria que ocupan los programas, y esto se nota sobre todo en los más largos.

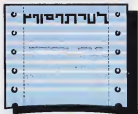
Pero no todo iba a ser un camino de rosas, también hay inconvenientes. En caso de error de programación nos costará hallarlo, puesto que no tenemos mensajes de error, que siempre nos pueden ayudar a solucionar los problemas. Otra característica a apuntar es que aunque cada orde-



Para el ordenador todos los lenguajes son iguales, ya que se reducen al código máquina.

ador utilice BASIC's diferentes, con muy poco conocimiento de las particularidades de la máquina podemos adaptar programas para que funcionen en diferentes «micro», sin embargo esto no es posible con el lenguaje máquina, porque nos obliga a conocer a fondo el funcionamiento

En la ejecución de un programa se pierde mucho tiempo en la traducción.

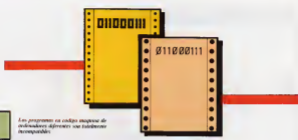


! !

Cada lenguaje ha sido creado para un uso concreto, pero todos, en el fondo, se reducen a uno: el lenguaje máquina.

\*

El proceso de traducción de un lenguaje de alto nivel es el que retrasa la ejecución de un programa, no tanto lo suficientemente rápido para ciertas aplicaciones.



Los programas en código máquina de ordenadores diferentes son totalmente incompatibles.

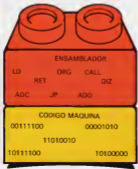
Los microprocesadores en la ausencia de error la falta de compatibilidad entre diferentes sistemas, dificultades de realización de los programas y mayor cantidad de instrucciones.

\*  
El ENSAMBLADOR solo «sabe» realizar sumas y restas.

\*  
El ENSAMBLADOR es un lenguaje de ayuda a la programación en código máquina de bajo nivel porque cada mnemónico es equivalente a una instrucción en el código máquina.

del microprocesador siendo mucho más trabajo. Los programas son difíciles de realizar porque cuesta más memorizar una lista de ceros y unos que unas palabras relacionadas con lo que queremos hacer (así 00111100 se olvida más fácilmente que LET A = A + 1), pero todavía se complica más si decimos que en el código máquina hay más instrucciones que en un lenguaje de alto nivel. Sin embargo podemos esta solucionar el proble-

*El ENSAMBLADOR es un lenguaje que tiene unos comandos que nos recuerdan qué función realizan y que se corresponden directamente con una instrucción del código máquina.*



ma de programar con ceros y unos si tuviéramos comandos que nos recordasen la función que realizan. Esto es lo que nos aporta el lenguaje EN SAMBLADOR.

## EL ENSAMBLADOR

Si a cada una de los 200 instrucciones del código máquina (formados todos por ocho unos y ceros) las asociásemos un símbolo que nos recordase la función que realizan y que fuera equivalente a cada comando del lenguaje máquina, nos costaría menos programar. Este conjunto de símbolos (que a partir de ahora llamaremos mnemónicos) es el que forma el lenguaje ENSAMBLADOR. Como último detalle, añadir que el mismo ENSAMBLADOR se encargará de realizar la traducción de los mnemónicos a los comandos del código máquina.

El ensamblador es un lenguaje de ayuda a la programación en código máquina de bajo nivel, porque cada comando, o mnemónico, es equivalente a una instrucción en el código máquina, y aunque sea necesaria una traducción, debido a la equivalencia que presenta apenas se nota una pérdida de velocidad. El lenguaje ensamblador aporta los mismos ventajas e inconvenientes que el código máquina, salvo la facilidad de recordar los mnemónicos, que aunque no son los comandos de los lenguajes de alto nivel, son mucho mejores que las series de números del código máquina.



Entonces, ¿será siempre aconsejable programar en ENSAMBLADOR? Es evidente que aunque nos facilite el trabajo no todo es tan sencillo como en el BASIC, por lo tanto hay que pensarse si en nuestro programa es conveniente utilizarlo o no. El ENSAMBLADOR (y también el código máquina) es bueno para todo aquello que necesite velocidad, por ejemplo para los juegos de arcade (los tipos de masacar marcianos), para manejar pantallas (cambios de paisaje en un juego), trabajar con ficheros, al ser posible encontrar un dato de forma muy rápida, para instrucciones que se repitan muy frecuentemente, al igual que para todo lo que necesite ahorro de memoria, como en los programas largos o chicos. Por el contrario, no es nada aconsejable para todo aquellas cosas simples, como la impresión de mensajes por pantalla, puesto que en estos casos no se aprecia ninguna ventaja al hacerlo por BASIC o ENSAMBLADOR.

No existe una fórmula mágica para decidir a cual lenguaje programar, aquí hemos pretendido dar unas ideas, pero es el programador el que tiene la última palabra dependiendo del tiempo disponible para trabajar (si el proceso es más largo, evidentemente, en el ensamblador), de la memoria que disponga nuestra máquina, la velocidad requerida, las necesidades que pretendamos satisfacer y otros factores que determinemos. Sin embargo, no es necesario realizar un programa entero en ENSAMBLADOR dado que podemos combinarlo perfectamente con el BASIC, y ahorrarnos el esfuerzo de «ensamblar» aspectos no importantes.

## PROFUNDIZAMOS

Ahora ya nos animamos más a programar en código máquina, porque nosotros lo hacemos en



CODIGO MAQUINA



BASIC



CODIGO MAQUINA

*El lenguaje máquina es bastante más rápido que el BASIC.*

ENSAMBLADOR y el mismo se encargará de pasarlo traducido a la máquina.

Sin embargo, para ello necesitamos que nuestra máquina tenga posibilidad de incorporar un ensamblador. Eso no es problema para el SPECTRUM que dispone de muchas versiones. Pero para tener una idea general de cómo es este lenguaje nos basta la versión de ARTIC COMPUTING comercializada por INDESCOMP.

Antes de continuar adelante es necesario diferenciar lo que es un «programa ensamblador» o un «programa en ensamblador». Lo primero se refiere al lenguaje en sí, es decir al programa encargado de traducir los mnemónicos a código máquina. El segundo es el programa que nosotros confeccionaremos, que será posteriormente «ensamblado» por el ENSAMBLADOR.

También debemos diferenciar entre lo que es el código fuente y el código objeto. El primero es el programa que escribimos lenguaje ensamblador, el segundo es el mismo programa traducido a lenguaje máquina por el ensamblador.

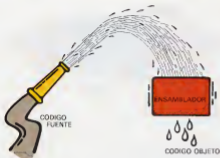
*En el código máquina tenemos más instrucciones que en un lenguaje de alto nivel.*



Las ventajas que nos ofrece el código máquina es la de su velocidad, hasta 60 veces más rápida que el BASIC, y el ahorro de memoria.



El ENSAMBLADOR presenta las mismas ventajas e inconvenientes que el código máquina, salvo la facilidad de recordar los mnemónicos.



*El código fuente es el programa escrito por nosotros, y el código objeto es cuando este traductor o lenguaje máquina.*

En este ensamblador el código fuente es situado automáticamente en una línea REM con el número 2. El código objeto lo hace también en una REM en la línea 1, a no ser que voluntariamente decidamos cambiar su emplazamiento con ORG. Por tanto, para grabar ambos códigos se hace como si fuera un programa BASIC y no como

*Registros es lo que usa el ENSAMBLADOR para guardar los datos*



bytes, que es lo que hacen otros ensambladores. Unos consejos importantes:

— En los programas largos grabar el código fuente y hacerlo por partes, para evitar disgustos cuando tengamos casi todo el programa tecleado (por ejemplo que nos quedemos sin luz)

— Antes de ejecutar un programa grabarlo. Si hay un error y se bloquea la máquina nos evitaremos teclear todo de nuevo.

— Cuando todo funcione ya se puede borrar el código fuente (REM línea 2 en el programa de ARTIC), aunque conviene guardar una copia de él para posibles rectificaciones posteriores.

Una de las diferencias entre el BASIC y el ENSAMBLADOR es que este no utiliza variables. Sus «variables» son llamadas registros, y sirven para guardar la información provisionalmente. Es en ellas donde se realizan las operaciones aritméticas y lógicas. Por tanto, cada vez que se use la palabra registro debemos pensar en un espacio en la memoria para guardar la información. Una vez cargado el ENSAMBLADOR estaremos directamente en el modo de inserción, pudiendo ya empezar a escribir nuestro programa. En caso de haber cometido algún error al escribirlo debemos entrar en el Editor para corregirlo. Para ello se pulsa E seguido de ENTER y tendremos acceso al menú que nos servirá para solucionar los problemas.

## LOS PRIMEROS COMANDOS



Un «programa ensamblador» es el encargado de traducir los comandos o códigos máquina. Un «programa en ensamblador» es el que nosotros hacemos en este lenguaje.



Código fuente es el programa escrito por nosotros en ENSAMBLADOR. Código objeto es el mismo programa traducido (o ensamblado) a código máquina.

Ya a estas alturas nos hemos podido percatar que el ensamblador es un lenguaje más, aunque tenga sus rarezas (todos los lenguajes las tienen). Los memorandos en ENSAMBLADOR realizan operaciones más simples, por eso es un lenguaje de bajo nivel, traducido directamente al código máquina, pero por el contrario ejecutable a una mayor velocidad. Otra particularidad que presentan es que tienen capacidad para manejar directamente direcciones de memoria, almacenando datos en donde consideramos oportuno, ante lo cual debemos tomar precauciones de dónde guardemos para que no se nos pierda nada por el intrincado bosque de las celdas de memoria. El ensamblador de INVESTRONICA tiene la particularidad de preferir los números hexadecimales a los decimales. Para los primeros no hay que especificar nada, mientras que para los segundos se antepone el signo «H» o «h» para que sepa que ese número es decimal.

Una de las instrucciones más similares a los lenguajes de alto nivel es LD, cuya función es la de asignación (el LET del BASIC). Su forma es:

LD A,2

Aquí guardamos el valor «2» en el registro «A» (recordemos que no es lo mismo un registro que una variable, aunque, sin que se entienda un infameñico, lo consideraríamos equivalente).

Una variante del mismo comando es:

LD (HL),2

En esta ocasión guardamos el dato (recordemos que está en hexadecimal) dentro de la dirección de memoria indicada en el registro «HL». Es decir, en este registro tenemos almacenado un número al cual nos indicará el número de celda de memoria en que debemos guardar el dato que se encuentra a continuación. Creo que no ofrecerá ningún problema lo siguiente:

LD A,(HL)

¿Lo traducimos? Ahora guardaremos en el registro «A» el valor que se encuentra en la dirección de memoria indicada por «HL». Por tanto cada vez

*El lenguaje máquina permite también la lectura del estado de memoria.*

que utilizemos «()» se hará referencia a la dirección de memoria y no al valor almacenado. La instrucción CALL supone un salto en la ejecución normal del programa, ya que éste se continuará ejecutando en la dirección de memoria a la

*Debemos analizar lo que vamos a programar para saber si resulta rentable hacerlo en ENSAMBLADOR.*



**i!**

Siempre que utilizemos un registro entre «()» se refiere a la dirección de memoria indicada por el valor que tenga el registro dentro de «()».

\*

Los movimientos son muy operaciones más simples pero más lentas y hacen la fealdad de manejar directamente direcciones de memoria.

\*

El ENSAMBLADOR no utiliza variables. Se equivalente sus los registros en donde se realizan todas las operaciones aritméticas y lógicas.



5 AHORRO DE MEMORIA



4 REPETICION DE INSTRUCCIONES



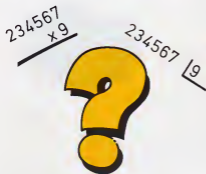
3 BUSQUEDA EN FICHEROS



2 MANEJO DE PANTALLAS



1 VELOCIDAD



El ensamblador no «sabe» realizar estas operaciones diferentes a sumar y restar

década a continuación. Nos sirve para a partir de dicha dirección, colocar una subrutina. Tenemos también la orden contraria, RET, que retorna el punto al que ha sido llamado por una instrucción CALL o retorna al BASIC si está llamado se ha producido desde éste (con el comando RAN-  
**DONIZE USR ...**), y, si es así, no nos debemos olvi-

Debemos tener cuidado en que sólo de nosotros hemos guardado los datos para recuperarlos cuando haga falta.



dar colocar este comando al final de nuestro programa en ensamblador para evitar que se nos quede «colgado» y no podamos recuperar el control del programa

---

---

## LA PILA

---

---

En informática una pila es una forma de guardar provisionalmente unos datos numéricos. En el caso del ENSAMBLADOR se puede simular a esas archiverías que tienen un cierre largo donde se pinchan los papeles. Si clavamos varios papeles, el último que hayamos metido será el primero que podemos sacar (de una manera normal, se supone). A este tipo de pila se la conoce como LIFO (Last In, First Out), último dato en entrar, primero en salir.

El ENSAMBLADOR utiliza la pila (también conocida como atacé) para almacenar datos de una forma provisional (no siendo posible realizar ninguna operación en ella. La única finalidad que tiene es guardar números cuando tenemos todos los registros ocupados).

Para trabajar con la pila tenemos dos comandos. El primero de ellos es PUSH (¡poner!), que no nos sorprenderá al conocer su misión: guardar la información en la pila. Así, PUSH AF almacenará en la pila el valor que se encuentre en el registro «AF».

Pero no nos serviría de nada si la información fuera irrecuperable. Para ello disponemos de POP (¡que nos suena a «sacar» el tapón de una botella de vino!). El mecanismo es igual que en la orden anterior: saca el valor de la pila, el superior (acuérdese del ejemplo del pincho de papeles, sólo podemos extraer el último introducido), y lo almacena en el registro indicado: POP AF.

---

---

## INCREMENTAMOS Y DECREMENTAMOS

---

---

En ENSAMBLADOR disponemos de la facilidad de unos contadores que aumentarán o disminu-





Una pila es un medio para guardar datos sucesivos. El último dato es el que está en la pila y el primero en la base.

ción del valor de lo contenido en el registro al que se haga referencia en una unidad.

Aquí seguimos contando con la ventaja que aporta el ENSAMBLADOR al recordar la escritura de los comandos la función que realizan.

Así, **INC** «incrementa» en 1 el valor del dato que se encuentra en el registro indicado. Veamos el siguiente ejemplo: **INC HL**. El contenido del registro HL es aumentado en uno tras ejecutar este comando.

También podemos «decrementar» el valor de un registro en una unidad. La sintaxis y el significado no presentarán ningún problema: **DEC HL**. Podemos complicar todavía más la situación con un salto condicional combinado con un decremento, y todo esto en un solo comando.

Para ello tenemos **DJNZ**, cuya función cada vez que se ejecuta es ¿preparados? decrementar el valor que se encuentra en el registro «B» y si no es cero se ejecutará lo que se encuentre a continuación del comando, y si lo es no habrá salto y seguirá por las instrucciones que se encuentran debajo.

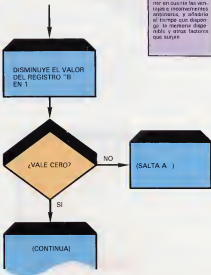
Tenemos también, como en todos los lenguajes, los saltos incondicionales. El representante más simple es **JMP** (del inglés *jump*). Por ejemplo, **JMP (HL)**. Acabemos de producir un salto en la ejecu-

ción del programa a la celda de memoria indicada por el valor que se encuentra en «HL».

## SUMAMOS Y RESTAMOS

No nos llevaremos ninguna sorpresa si le decimos que con el ensamblador podemos sumar y restar. Aunque al parecer si le decimos que solo «sabe» realizar estas dos operaciones, y en realidad no necesita más, porque se pueden ha-

cer de ver la función de **DJNZ**.

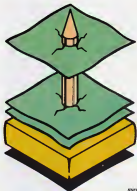


!

El salto condicional puede estar representado por **DJNZ** que decrementa el valor del registro «B» en uno y si no es cero salta a donde se lo indiquen y si lo es continúa la ejecución del programa.

\*

La última palabra sobre cual lenguaje utilizar la tiene el programador, que deberá tener en cuenta las ventajas e inconvenientes asociados, y además el tiempo que disponga, la memoria disponible y otros factores que surjan.



"POP"

"PUSH"

PUSH envía un dato en la pila, mientras POP extrae el último dato guardado, el de arriba.

Por los demás como combinación de sumas y restas. Para sumar haremos **ADD A,H**.

En el ejemplo se produce la suma (**ADD** de «adición») del número contenido en los registros «A» y «H» guardando el valor en el primero. Si operamos con el registro «A» no es necesario especificarlo, escribiendo solamente **ADD (HL)**. En este caso se suma los valores de «A» y del con-

tenido en la dirección de memoria especificada por «HL», guardando el resultado en «A». De forma parecida opera la resta, su comando es **SUB** (de «substracción», y funciona igual que la suma **SUB A,H**).

## COMENZAMOS

Nuestro objetivo al tratar el tema de este lenguaje es quitar el miedo que existe con el código máquina y con el **ENSAMBLADOR**, e intentar presentarlo como lo que es: un lenguaje más, necesitando para ello un programa ensamblador como el aquí comentado, a pesar, en este caso, que por culpa de las «man»-instrucciones-que lo acompaña se haga cuesta arriba. Como es normal, la guía no enseña el lenguaje, sólo explica las características y peculiaridades que presenta, pero sin muchos detalles.

Esperemos que ese caja negra que es para muchos el **ENSAMBLADOR**, haya desaparecido, y se anime a hacer sus primeros pasos, no conformándose con lo aquí expuesto, sino aprovechar este final para comenzar a «ensamblar».



**INC** aumenta en uno el valor de la contenido en el registro y **DEC** disminuye dicho valor en uno.



# MICROARITMETICA



La ALU (unidad aritmética y lógica) situada en el interior del Z 80 es la encargada de gestionar, entre otros trabajos, aquellos que impliquen la ejecución de sumas, restas y manejo de los operadores lógicos AND, OR y XOR.

Estas últimas ya fueron descritas en el capítulo

anterior, por lo que en este nos centraremos en las instrucciones aritméticas previstas por los fabricantes del microprocesador instalado en nuestro Spectrum.

Grupo aritmético de 16 bits

Mnemónico	Código Máquina	REGISTRO R								Nº BYTES	CICLOS		NOTAS
		7	6	5	4	3	2	1	0		MAD	RELOJ	
		S	Z	H	OV	N	C						
ADD HL, ss	0 0 0 0 1 0 0 1	•	•	•	•	•	•	•	•	1	3	11	Reg ss
ADC HL, ss	1 1 1 0 1 1 0 1	•	•	•	•	•	•	•	•	2	4	10	BC 00 DE 01 HL 10 SP 11
SBC HL, ss	1 1 1 0 1 1 0 1	•	•	•	•	•	•	•	•	2	4	15	Reg PP
ADD IX, pp	0 0 0 0 1 0 0 1	•	•	•	•	•	•	•	•	2	4	15	BC 00 DE 01 IX 10 SP 11
ADD IY, rr	0 0 1 1 1 0 0 1	•	•	•	•	•	•	•	•	2	4	15	
INC ss	0 0 0 0 0 0 1 1	•	•	•	•	•	•	•	•	1	1	6	Reg rr
INC IX	0 0 1 0 0 0 1 1	•	•	•	•	•	•	•	•	2	2	10	BC 00 DE 01 IY 10 SP 11
INC IY	0 0 1 0 0 0 1 1	•	•	•	•	•	•	•	•	2	2	10	
DEC ss	0 0 0 0 1 0 1 1	•	•	•	•	•	•	•	•	1	1	6	
DEC IX	0 0 1 0 1 0 1 1	•	•	•	•	•	•	•	•	2	2	10	
DEC IY	0 0 1 0 1 0 1 1	•	•	•	•	•	•	•	•	2	2	10	

## GRUPO ARITMETICO DE 8 BITS

Pertenecen a este grupo, todas aquellas instrucciones que efectúen la suma (ADD) o resta (SUB)

de un valor contenido en un registro o posición de memoria con el almacenado en el acumulador, quedando el resultado en esta última y manteniéndose inalterado el valor del registro o posición de memoria implicado tras la operación aritmética.

Generalmente podríamos representar estas operaciones de la manera siguiente:

SUMA      A+n → A  
RESTA      A-n → A

MNEMONICO	CODIGO MAQUINA	REGISTRO F								Nº BYTES	DECOD.		NOTAS														
		7	6	5	4	3	2	1	0		MAD	RELOJ															
		S	Z	H	PV	M	C																				
ADD r	1 0 0 0 0 0 r r r	0	0	0	0	0	0	0	0	1	1	4	<div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;">                     Reg                 </div> <table style="font-size: small; border-collapse: collapse;"> <tr><td>A</td><td>111</td></tr> <tr><td>B</td><td>000</td></tr> <tr><td>C</td><td>001</td></tr> <tr><td>D</td><td>010</td></tr> <tr><td>E</td><td>011</td></tr> <tr><td>H</td><td>100</td></tr> <tr><td>L</td><td>101</td></tr> </table> <div style="border: 1px solid black; padding: 5px; margin-top: 5px; text-align: center;">                     S                      r                      (M)                      (X+)                      (Y+)                 </div>	A	111	B	000	C	001	D	010	E	011	H	100	L	101
A	111																										
B	000																										
C	001																										
D	010																										
E	011																										
H	100																										
L	101																										
ADD n	1 0 0 0 0 1 1 0	0	0	0	0	0	1	1	0	2	2	7															
ADD (HL)	1 0 0 0 0 1 1 0	0	0	0	0	0	1	1	0	1	1	4															
ADD (X+)	1 1 0 1 1 1 0 1	1	1	0	1	1	1	0	1	3	6	19															
ADD (Y+)	1 1 1 1 1 1 0 1	1	1	1	1	1	1	0	1	3	6	19															
ADC s	0 0 1 1 1 1 1 1	0	0	1	1	1	1	1	1	04	04	04															
SUB s	0 1 0 1 1 1 1 1	0	1	0	1	1	1	1	1																		
SBC s	0 1 1 1 1 1 1 1	0	1	1	1	1	1	1	1																		
INC r	0 0 1 r r r 1 0 0	0	0	1	r	r	r	1	0	1	1	4															
INC (HL)	0 0 1 1 0 1 0 0	0	0	1	1	0	1	0	0	1	3	11															
INC (X+)	1 1 0 1 1 1 0 1	1	1	0	1	1	1	0	1	3	6	23															
INC (Y+)	1 1 1 1 1 1 0 1	1	1	1	1	1	1	0	1	3	6	23															
DEC s	0 0 1 1 0 1 0 1	0	0	1	1	0	1	0	1																		

- Indicador de afectado
- Indicador modificado según el resultado de la operación
- P El bit 2 del registro F actúa como indicador de paridad (1 - paridad par/O - paridad impar)
- V El bit 2 del registro F actúa como indicador de sobrepasamiento (1 - sobrepasam / 0 - no sobrepasam)
- X Estado indeterminado
- (r) (R) (c) En las instrucciones donde se utiliza la abreviatura "r" el nº de bytes es los registros y de reloj son las retransmisiones correspondientes originales

\* Solo en DECd

donde A representa el valor almacenado en el acumulador, y n el contenido del registro o posición de memoria especificado en la instrucción. Además, es posible incrementar (INC) o decrementar (DEC) en 1 la cantidad almacenada en un determinado registro. Veamos el efecto que sobre los indicadores tienen estas operaciones:

S—Cuando el resultado en A es negativo, es decir, si el bit más significativo de este es 1, la bandera de signo contendrá un 1. En caso contrario será 0.  
Z—Tras una operación aritmética, su valor será 1, si el resultado de ésta fue 0. De lo contrario, contendrá un 0.

N—El indicador de resta se coloca a 1 en todas las operaciones relacionadas con esta operación, mientras que en las sumas permanece a 0.

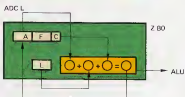
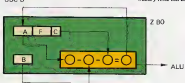
P/V—Señala si hubo sobrepasamiento (1) o no (0) en las operaciones aritméticas de dos números expresados en complemento a dos.

C—Actúa de manera similar al anterior, pero con números expresados en su correspondiente formato binario.

H—Su importancia se pone de manifiesto cuando ejecutamos operaciones aritméticas con números en formato BCD (decimal codificado en binario), los cuales serán discutidos próximamente, al analizar la instrucción DAA.

En el capítulo dedicado a la presentación de las tablas de instrucciones, se señalan los operandos sobre los cuales actúan las de memoria ADD, SUB, INC y DEC. Como podemos comprobar, además están contempladas dos operaciones especiales para la suma y la resta, las cuales tienen presente el valor de la bandera de acarreo C. Se trata de aquellas cuyo mnemónico es ADC (sumar con acarreo) y SBC (restar con acarreo).

SBC B Suma y resta con acarreo



Implementar estas instrucciones obliga al microprocesador a seguir los siguientes pasos basados en el esquema:

SUMA con acarreo  $A+n+C \rightarrow A$

RESTA con acarreo  $A-n-C \rightarrow A$

1) Suma o resta al/del acumulador el byte especificado en la instrucción como si se tratara, al

MNEMÓNICO	CÓDIGO MAQUINA	REGISTRO R								NP BYTES	CICLOS		NOTAS
		7	6	5	4	3	2	1	0		MAG	PELOJ	
		S	Z	H	P/V	N	C						
DAA	0 0 1 0 0 1 1 1									1	1	4	
DPI	0 0 1 0 1 1 1 1									1	1	4	
NEG	1 1 1 0 1 1 0 1									2	2	8	
DCF	0 0 1 1 1 1 1 1									1	1	4	
SCF	0 0 1 1 0 1 1 1									1	1	4	
NOP	0 0 0 0 0 0 0 0									1	1	4	
HALT	0 1 1 1 0 1 1 0									1	1	4	

Grupo anterior de la CPU.

RESTANDO EN EL Z 80

$$\begin{array}{r}
 01001000 \quad \leftarrow \text{MINUENDO} \quad 72 \\
 -00011100 \quad \leftarrow \text{SUSTRAYENDO} \\
 \hline
 \end{array}$$

1.—COMPLEMENTO A 2 DEL SUSTRAYENDO

$$\begin{array}{r}
 11 \text{ COMPLEMENTO A 1} \rightarrow 11100011 \\
 12 \text{ SUMO 1} \rightarrow +00000001 \\
 \hline
 11100010
 \end{array}$$

2.—SUMO MINUENDO + COMPL. A 2 DEL SUSTRAYENDO

$$\begin{array}{r}
 01001000 \\
 +11100010 \\
 \hline
 00101010 \rightarrow 44 \text{ DECIMAL}
 \end{array}$$

Recordar es el Z 80

igual que antes, de ADD o SUB, pero sin modificar ningún bit del registro de indicadores.  
 2) El valor del bit de acarreo es sumado o restado al/del acumulador, es decir, si previamente a la ejecución de la instrucción estaba alzado (1), se suma o resta 1 de A.  
 3) Finalmente el Z 80 ajusta las banderas del registro F, basándose en el último resultado recibido en A.



EL GRUPO ARITMETICO DE 16 BITS

El BCD (Binary Coded Decimal) es un modo para codificar los números decimales empleando 4 dígitos binarios por cada uno decimal.



Cuando trabajamos en aritmética BCD tras cualquier operación debemos ejecutar una instrucción DAA para evitar posibles conflictos.

Las instrucciones aritméticas anteriormente mencionadas ADD, ADC, SBC, INC y DEC, que actúan sobre registros o posiciones de memoria de 8 bits, tienen sus análogas cuando se trata de operar con 16 bits, pero solamente trabajan con cantidades almacenadas en registros dobles, no siendo posible recogerlas de una celda de memoria determinada.

En el caso de las sumas y restas con 8 bits, el primer operando no era necesario especificarlo en la instrucción, pues el microprocesador tomaba siempre por defecto el valor almacenado en el acumulador.

Cuando se trata de sumar o restar números de 2 bytes, el primer operando ha de estar contenido en el par HL, tal y como en las instrucciones ADD DL,pp y ADD HL,r donde se toman los registros indicados con este fin.

El segundo operando será el valor almacenado en cualquier par de registros de los autorizados en la instrucción en concreto de que se trate (rs, pp o rt), siguiendo la decodificación expuesta en la tabla, sin que sea posible trabajar con parejas como BL o HC.

Tras efectuar la suma o resta, el resultado queda almacenado en HL (en IX o IY si es su caso), manteniéndose inalterado el contenido del par de registros utilizado como segundo operando.

Aclaremos su funcionamiento con algunos ejemplos: supongamos que deseamos ejecutar la instrucción ADD HL,DE siendo el valor contenido en HL 52277 y en DE 23758.

	ANTES	DESPUES
H	11001100	00101001
L	00110101	00000011
D	01011100	01011100
E	11001110	11001110

Es decir, tras sumar 52277 y 23758 (hemos obtenido como resultado 10499) en vez de 76035 como sería de esperar. Podríamos pensar que el Z 80 no tiene ni idea de lo que es una suma. Si nos fijamos en la tabla donde se detallan los indicadores afectados observaremos que sólo tres de ellos pueden haber cambiado.

La bandera de transcarro (H) se alzará en estas operaciones, cuando al ajustar la suma binaria nos vayamos «llevando una» desde el bit 11 hasta el final. En nuestro ejemplo no ocurre esto y por tanto permanecerá a 0. El indicador de resta N contendrá 1 cuando efectuemos, por ejemplo, una instrucción SBC, es decir, una resta. En nuestro caso la operación implementada fue una suma y contendrá 0, por tanto.

Hasta ahora no hemos conseguido detectar ninguna condición que permita justificar tan sorprendente resultado y sólo nos queda uno de los bits del registro F: el representativo del acarreo. Este bandera se alza cuando se produce acarreo en el bit 15 (si me «lleva una» a partir del bit 15 al realizar la suma) y eso precisamente es lo que ha ocurrido en nuestro ejemplo (comprobémoslo). Entonces el bit de acarreo C del registro F contendrá 1.

¿Cómo interpretamos entonces el resultado? Todos sabemos que con 16 bits podemos representar números decimales entre 0 y 65535, es decir 65536 combinaciones distintas. Pues 10499 es exactamente la cantidad resultante de efectuar la operación 52277+23758-65536.

Como el par HL sólo puede contener números de 16 bits, es decir, inferiores a 65535, el 76035 se le hace demasiado grande y por ello la resta 65536, circunstancia que detectamos al comprobar que el banderín de acarreo está alzado. Las instrucciones de incremento y decremento de 16 bits son completamente análogas en su funcionamiento a sus correspondientes de 8 bits, sólo que ahora actúan sobre registros dobles.

Al ejecutarse se suma/resta 1 del contenido del byte de menor peso dentro del par especificado en la instrucción. Si esta operación provoca acarreo, éste es transmitido automáticamente al octeto superior.

En los programas en C/M se utilizan para realizar bucles de más de 256 repeticiones, donde puede resultar interesante conocer cuándo hemos llegado, a base de decrementar un valor pre-fijado, a 0.

Como estas instrucciones no afectan al contenido del registro de indicadores podremos pensar que la anterior circunstancia no es detectable. Sin embargo, podemos sobrevivir del siguiente artificio: supongamos que hemos estado decrementando el par BC. Tras ejecutar las instrucciones LD A,B OR C

la bandera de cero Z contendrá un 1, solamente cuando tanto B como C sean 0.



## EL GRUPO ARITMETICO DE LA CPU

Pertenecen a éste algunas de las instrucciones incluidas en el capítulo 41 bajo la denominación «aritmética y de control de la CPU», las cuales vienen a complementar a las anteriores. Comentemos aquí lo exactamente su cometido:

— CPL (ComPLement, complementar) efectúa el complemento a 1 del valor almacenado en el acumulador; es decir, cambia los 1 por 0 y los 0 por 1.

— NEG (NEGative, negativo) realiza la negación del acumulador, operación que ha de entenderse como encontrar su complemento a 2. En reali-

dad, el microprocesador lo que hace es restar de 0 el contenido de A (0-A), y para ello, como en todas las operaciones de resta complementa a 2 el sustruendo y le suma después el minuendo.

— CCF (Complement Carry Flag) complementa el valor actual de la bandera de acarreo C, es decir, si contiene 1, la coloca 0 y viceversa.

— SCF (Set Carry Flag) pone a 1 el indicador de acarreo C.

— DAA (Decimal Adjustment Accumulator) se encarga de efectuar el ajuste decimal del acumulador tras una operación aritmética de dos números en formato BCD.

El BCD es un sistema de codificación empleado de representar cada dígito correspondiente a un número decimal mediante cuatro dígitos binarios. Por ejemplo, el número decimal 76 tendría el siguiente aspecto en BCD:

01100110

Si quisiéramos representar un número mayor, por ejemplo, 6582 necesitaríamos dos octetos 01000101 y 10010010.

Al realizar una operación aritmética con números en este formato, lo normal es que el resultado recogido en el acumulador no esté en BCD y para transformarlo a este, se utiliza la instrucción DAA, de manera que no se produzcan errores de interpretación entre cantidades en binario y BCD.

A la anteriores podemos añadir dos instrucciones, las cuales afectan directamente a la CPU. Se trata de NOP (No Operation) la cual simplemente indica al Z80 que no haga nada. Esto no quiere decir que ejecutar esta instrucción no tome tiempo, pues de hecho se requieren 4 ciclos de reloj para llevarla a cabo. Su utilidad versa en proporcionar espacio dentro de un programa al objeto de añadir luego otras instrucciones. Finalmente la orden HALT provoca que el microprocesador pare cualquier trabajo que esté efectuando.

Con esto cerramos el análisis de los grupos aritméticos y lógico del Z80 emprendido en el capítulo anterior. En el próximo discutiremos las instrucciones que provocan bifurcaciones dentro de la secuencia de un programa.

! !

La instrucción NOP precisa de 4 ciclos de reloj para completar su ejecución.

\*

Las operaciones aritméticas con números de 16 bits, sólo es posible efectuarlas cuando estas se encuentran situadas en registros de bytes.

\*

El resultado de una operación aritmética de 8 bits siempre está almacenado en el acumulador.

\*

HALT provoca la detención en cualquier trabajo que el microprocesador esté efectuando.



La A&U es la encargada de procesar las instrucciones aritméticas y lógicas.



# PALETA

!!



¿Cúen fue el estado que proclamo a los seis vientos que nuestro gran amigo Spectrum, sólo era capaz de generar ocho colores? Con este programa, nuestro pequeño ordenador no generará ocho, ni cuarenta, ni ochenta colores, sino ciento veintisiete. ¿Asombrados?

Para la grabación del programa emplearemos el comando **SAVE "PALETA"**, o bien, **SAVE "PALETA" LINE 10**, si de nosotros que esto se ejecuta automáticamente al finalizar su carga.

\*

Los caracteres A su localización en las líneas 1340 y 1320 del listado corresponden al gráfico definido de dicha tecla.

## EL PROGRAMA

Nuestro programa PALETA genera ciento diecisiete mezclas, partiendo de los ocho colores básicos con los que normalmente puede trabajar el Spectrum. Estos ciento diecisiete tonos, junto

Al ejecutar el programa aparece en la pantalla un cuadro con los siguientes tipos de colores, a los cuales podemos acceder mediante un código numérico

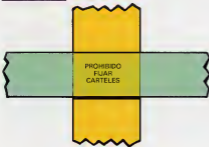


Nuestro programa PALETA genera 117 colores, partiendo de los 8 colores básicos del Spectrum.

con los ocho anteriormente mencionados forman el conjunto de ciento veintisiete colores. Tanto los tonos primarios como las mezclas, están formados por la combinación de la tinta y papel en un gráfico definido de usuario, concretamente en el U D G de «A». La configuración tanto binaria como decimal de dicho gráfico definido es la siguiente:

BINARIO	DECIMAL
10101010	170
01010101	85
10101010	170
01010101	85
10101010	170
01010101	85
10101010	170
01010101	85

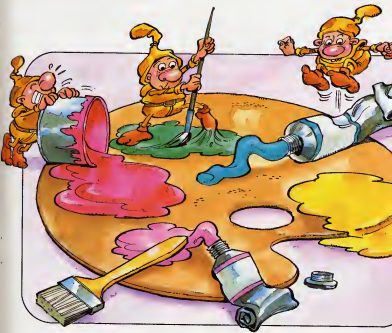
PROHIBIDO  
FUJAR  
CARTELES





Al asignar un color de tinta y otro de papel a este gráfico definido, el Spectrum actúa los bits a 1 con el color de la tinta elegida, y los bits a 0 con

el color del papel. Para nuestro ordenador sólo existen dos colores, pero nosotros, al poseer una percepción óptica más o menos compleja, fu-



señamos los dos tonos en uno solo, captando así la mezcla de los dos colores



Debemos tener en cuenta que algunas de las mezclas pueden ser utilizadas para escribir sobre ellas

Nada más ejecutar el programa, aparece en la pantalla un menú con los diferentes tipos de colores a los cuales podemos acceder mediante un código numérico situado a la izquierda del tono en cuestión

Este código numérico consta de dos cifras. La primera nos indica el color del papel y la segunda el de la tinta

En el menú de presentación existen solamente 64 colores. Por cada tono elegido, nuestro Spectrum generará dos diferentes: uno sin brillo (BRIGHT 0) y otro con el brillo activado (BRIGHT 1)

Sólomente existe un caso en donde el atributo «brillo» no ejerce ninguna influencia: cuando tanto el papel como la tinta son de color negro (papel 0, ink 0, código del programa=00)

Debemos tener muy en cuenta que ninguna de las mezclas que se obtienen de los ocho colores básicos no pueden ser utilizadas como «pantalla» escribir o mensajes encima de estas, ya que, a causa del hardware, el Spectrum sólo puede mantener a un mismo tiempo dos colores básicos en una posición de carácter.

Si nuestro deseo es encontrar nuevos tonos de mezclas, lo único que debemos hacer es reformar la definición del gráfico definido por otra diferente a la que ya existe en el programa. Así pues lo único que debemos cambiar es el contenido de la DATA en la última línea del programa

Ejemplo 1  
DATA 255,0,255,0,255,0,255,0

Ejemplo 2  
DATA 128,128,128,128,128,128,128,128,128

```

10 REM *****
20 REM * J. M. MAYORAL SERRANO *
30 REM *****
40 REM * PALETA (C) 1986 *
45 REM *****
50 POKE 23658,0
100 PAPER 7: INK 9: BORDER 7
150 CLS : PRINT INK 11: 127 COLORES PARA SPECTRU
M
160 REM [INV_] BRANCOS DE USUARIO [TRUE]
200 FOR N=0 TO 7
210 READ M
250 POKE USR 'A'+N,M
300 NEXT N
500 GO SUB 1700
510 REM [INV_]INTRO. CODIGO NUMERICO[TRUE]
750 INPUT , LINE A$
900 IF LEN A$<2 THEN GO TO 750
850 IF CODE A$<40 OR CODE A$>55 OR CODE A$(2)<40 OR
CODE A$(2)>55 THEN GO TO 750
870 REM [INV_] ASIGNACION VALORES [TRUE]
840 LET B=VAL A$(2)
865 LET A=VAL A$(1)
870 CLS : LET BRILLO=0
890 GO SUB 1900
890 LET BRILLO=1
895 GO SUB 1900
900 REM [INV_] MENSAJES EIN PROG. [TRUE]
960 IF VAL A$(1)>0 AND VAL A$(2)=0 THEN PRINT AT 0,
3:ESTE COLOR SE OBTIENE CON: : PRINT AT 11,10: ' EL C
OLOR ':A$(1):' : GO TO 1100
1000 PRINT AT 3,3:ESTE COLOR SE OBTIENE CON: : PRINT
AT 5,2:PRINT PAPER :A: : INK :B: :BRIGHT 0'
1010 PRINT AT 13,3:ESTE COLOR SE OBTIENE CON: : PRIN
T AT 15,2:PRINT PAPER :A: : INK :B: :BRIGHT 1'
1450 PRINT INK 2:AT 21,2: DESEAS CONTINUAR ? ' : I
NK 1: :S/N'
1460 LET K$=INKEY$
1500 IF K$=' ' THEN GO TO 1460
1550 IF INKEY$='S' THEN RUN
1600 IF INKEY$='N' THEN GO TO 10000
1650 GO TO 1500
1700 REM [INV_] PRESENTACION [TRUE]
1785 PRINT : PRINT
1710 FOR A=0 TO 7
1720 FOR B=0 TO 7
1730 PRINT A;B;
1740 PRINT PAPER A: INK B: 'AAAAAA';
1750 NEXT B
1760 NEXT A
1770 PRINT : PRINT
1780 PRINT 'INTRODUCE EL CODIGO NUMERICO DEL '
1790 PRINT ' DEL COLOR ELEGIDO'
1800 RETURN
1900 REM [INV_] RELLENO DE PANTALLA [TRUE]
1910 FOR N=0 TO 9
1920 PRINT PAPER A: INK B: BRIGHT BRILLO: 'AAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA'
1930 NEXT N
1940 RETURN
2000 DATA 178,85,178,85,178,85,178,85

```

